Research Diary
Kyle Newman V00781162

Week of Oct 2nd:

Assigned:

1. Check out http://www.ismir.net/. Look at templates for conferences for papers.
2. Create a working document (Research Diary). Document resources and dates accessed
3. Pour over article and write down any questions/vocabulary. Prepare a path to answer those questions yourself.
4. Search for datasets to train NN's.

Goals for next week:

1. Research MIDI in more detail
2. Research NN's in more detail (Tutorials online)
3. Distill information in the research article (Article Summary)
4. Start a bibliography.

Vocabulary:

Acoustic Model: Will be used for the for determining and classifying the acoustic properties of the signal. MLM will handle the musical probabilities, AM will handle the pitches themselves.
Music Language Model: In speech recognition, the acoustic signal isn't enough to differentiate words due to homophones, dialects etc. According to this article, a language model is used to weight the probabilities between ambiguous words (ex. I want to go ___. **There** is more likely than **their**). Similarly, music follows a somewhat predictable pattern, so a model can be used to resolve ambiguities and predict notes.

Temporal Structure: can be quantified in terms of the predictability of the changes in a given property over time. [1] i.e. structure built through time.

Spatial Structure: structure built through space.

Network Types – Acoustic Model:
Recurrent Neural Network: NN that is suited for sequential data. Characterized by recursive connections between layers. Since MLM has a temporal structure, RNN's are suited for this model.

- **Back Propogation Through Time Algorithm (BPTT):** unfolds the RNN into a feedforward NN s.t. we can apply the Backpropagation Algorithm[2].

(Bi-directional RNN's?)
Deep Neural Network: Classic NN. Good for classification and static tasks. Has one or more non-linear transformations (in our case, the sigmoid fn).

- **Posterior Probability Distribution:** description of a random phenomenon in terms of probabilities/events, applied after the evidence has been taken into account.
- **Constant Q Transform:** An STFT will tell you how much info is stored at each frequency (spectrogram). CQT a spectrogram with logarithmic spaced frequencies.
- **Backpropogation Algorithm:** See BPTT above.

1 http://www.psy.vanderbilt.edu/faculty/blake/TS/TS.html
2 https://page.mi.fu-berlin.de/rojas/neural/chapter/K7.pdf, page 24

Convolution Neural Network: used in image processing. Attempt to mirror the visual cortex of animals. AKA space invariant NN's[3].

- **Feature Map:** a hidden layer is segmented into feature maps where each unit in a feature map looks for the same feature, just at different positions[4].
- **Four Dimensional Tensor:**
- **Max Pooling Function:**

Features & Classifiers (for NN's)

Network Types – M Language Model:
Generative RNN:
Neural Autogressive Distribution Estimator: Models distributions of high-dimensional vectors. Inspired by the restricted Boltzmann maching (RBM)[5].
RNN-NADE (combo): "Sequence of NADE's conditioned on an RNN, that describe a distribution over sequences of polyphonic music" (page 4).

Hybrid RNN:
Emission Probability: Output probabilities
Gradient Descent: optimized algorithm used to find a local minimum in a function.
Dropout Rate: A method to keep DNN from overfitting. A problem for NN's with a large number of parameters[6].

Inference Model:
Beam Search: at time t, BS holds (w = beam width) partial solutions. Solutions in the beam at t correspond to sub-sequences of length t. Next, all possible descendants of the w partial solutions in the beam are enumerated and then sorted in decreasing order of log-likelihood. From these candidate solutions, the top w solutions are retained as beam entries for further search.

Beam search is used for searching for candidates when the number of solutions is limited, but since polyphonic music has that large output space, it is too inefficient. They propose constraining the amount of candidates being considered by a *branching factor* K.

Frame Based Metrics:
Note Base Metrics:

Dynamic Bayesian Network[7]

Algorithms:
Expectation Maximization (EM) Algorithm: iterative method for finding maximum likelihood of parameters in statistical models.
Beam Search Algorithm: See above
Hashed Beam Search: See above
Backpropogation Algorithm: See BPTT above.
Back Propogation Through Time Algorithm (BPTT): unfolds the RNN into a feedforward NN s.t. we can apply the Backpropagation Algorithm.

3 http://deeplearning.net/tutorial/lenet.html
4 https://www.youtube.com/watch?v=aAT1t9p7ShM
5 http://jmlr.csail.mit.edu/proceedings/papers/v15/larochelle11a/larochelle11a.pdf
6 https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf
7 http://mlg.eng.cam.ac.uk/zoubin/papers/ijprai.pdf (HMM and Bayesian Networks)

Equivalent Rectangular Bandwidth (ERB) filterbank:
Gradient Descent: optimized algorithm used to find a local minimum in a function.
Stochastic Gradient Descent: Stochastic attempts to iteratively find zeroes or extremes of functions which can't be computed directly.

Math and Models:
Basis Spectra: (guess) a mapping of the spectrogram into pitches (page 1). The frequencies that form a piano note, form a basis.
Non-Negative Matrix Factorization (NMF): Factorization of a positive matrix into two positive matrices.
Hidden Markov Models: A type of recurrent network[8].
Probabilistic latent variable component analysis (PCLA): fits a latent variable probabilistic model to normalized spectrogram.
Support Vector Machines (SVM): Supervised learning algorithms[9].
Deep Belief Network (DBN): A type of DNN, made of connected layers and unconnected units within those layers[10].
Baye's Rule: $P(A \mid B) = [\ P(B \mid A) * P(A)\ ] / P(B)$ :: if cancer is related to age, then, using Bayes' theorem, a person's age can be used to more accurately assess the probability that they have cancer.
F-Measure: = 2 * (precision * recall) / (precision + recall). Used to measure a test's accuracy.
Tensor[11]: geometric objects that describe linear relations (ex. Dot products, linear maps, etc.)

Questions:

Polyphony has a combinatorialy large output space.
This makes sense, but how large is it?
The article says it's larger than Speech. How large is speech?

The non-linear function used in this article's NN's is the sigmoid function. Why? From wolfram, it looks linear on a local level, and non-linear in the extremes? The sigmoid function seems to be a used as a relaxation of the step function used in perceptrons. The sigmoid is a popular non-linear function for DNN's[12].

In section D, the network training involves the "hyperbolic tangent function." What does that do? The tanh(n) function is similar in shape to the sigmoid function, but has asymptotes at [-1, 1] rather than the sigmoid's [0,1][13],[14].

In the Hybrid RNN model, how does independent training of the two Models offset the issue of small MIR datasets? (page 5).

What is an end-to-end architecture? An end to end network is one where application-specific features (important info) resides only in the ends of the framework, rather than at gateways within the

8 https://page.mi.fu-berlin.de/rojas/neural/chapter/K7.pdf, page 27
9 http://www.cs.columbia.edu/~kathy/cs4701/documents/jason_svm_tutorial.pdf
10 http://deeplearning.net/tutorial/DBN.html
11 https://en.wikipedia.org/wiki/Tensor
12 https://page.mi.fu-berlin.de/rojas/neural/chapter/K7.pdf, page 2
13 https://page.mi.fu-berlin.de/rojas/neural/chapter/K7.pdf, page 2, 3
14 http://ufldl.stanford.edu/wiki/index.php/Neural_Networks

network. For two processes communicating with each other, the reliability cannot be expected to be aligned with the reliability of the processes. Obtaining reliability in the communication of the processes is more costly than obtaining it at the ends of the network.

***** Future work section mentions the system's trouble with sustain. Longer notes are cut short. The article recommends adding samples with noise and deformations,**

***** Future work also recommends replacing the CQT with an ERB or a variable-Q transform.**

$\propto$ **:** Means "varies as" or "proportional to"
Π : The product version of a summation ($\sum$).

Datasets:

http://resources.mpi-inf.mpg.de/SMD/ :: Saarland Music Data. Provides 50 midi – audio pairs for training networks.
http://www.tsi.telecom-paristech.fr/aao/en/2010/07/08/maps-database-a-piano-database-for-multipitch-estimation-and-automatic-transcription-of-music/ :: MAPS database. Used by the article in question. Contains 31 GB worth of .wav recordings. Around 270 recordings were used to train and test the system.
http://extras.springer.com/2013/978-1-4614-7475-3 :: database of chords (16.9 GB)

https://www.reddit.com/r/datasets/comments/3akhxy/the_largest_midi_collection_on_the_internet/ :: **Unexplored**. A mega.co.nz dump of 130,000 (3.65 GB) of MIDI files.

Nottingham Music Database http://abc.sourceforge.net/NMD/

http://www.piano-midi.de/ gives mp3 – midi pairs.

Week of Oct 9th

Summary of “An End-to-End Neural Network...”

1. Introduction:

The article begins by introducing the problem of AMT and its difficulties. The reason polyphonic AMT is so difficult is because overlapping notes can be ambiguous to classify. If you have two concurrent sounds, possible with different timbre/spectrums, it’s difficult to separate them. Furthermore, the total combinations of pitches (combinatorial space) is very large in practise.

We then go on to describe current methods of approaching AMT. The main problem of AMT centres around representing the input waveform (via its magnitude spectrum) as musical pitches. One method is done using NMF by learning a dictionary of pitches by unsupervised training. This approach is extended using PLCA which attaches a probabilistic model to the normalized spectrogram. PLCA and the EM algorithm seem to be the current approach to spectrogram factorization techniques.

The second approach to AMT is to use discriminative techniques[15] by relating features (recognizing possible pitches in the spectrum) from the audio and classifying it into pitches. The advantage is that instead of building something that models a specific instrument, we can build complex classifiers that identify pitches regardless of timbre. An example is to use SVM’s as a classifier for the input, and a DBN (type of DNN) to learn the features. Another example is to use an RNN on the mag spectrum for polyphonic music transcription.

The author then describes how speech recognition addresses these problems. Often there is an acoustic model to classify the sounds, and a language model to resolve ambiguities. For example, “I want to go there,” **I** and **eye** are homophones, but I is more likely to start a sentences. Similarly **there** and **their** are homophones, but there is more likely to fit in the sentence. Since music has a temporal structure, a language model could be useful for solving ambiguities (TS means that music is comprehended over time. Just as a sentence is more than just isolated words, a musical phrase is more than just isolated pitches). Usually, the MLM problem is instead handled with a two-state HMM which has also been extended to DBN’s in NMF techniques. Also, there is a PLCA-RNN combo, and a RNN only model (using a sequence transduction framework).

Finally, the author proposes their end-to-end architecture to train and combine the AM and MLM necessary for AMT, how they plan on evaluating, as well as exploring ConvNets as a possible AM.

2 Backgrounds:

How does a neural network work[16][17]?

Suppose we want to predict an output y from some input x. Ex. **Predict** score y (int) based on hours slept and hours studied (time, time). This is a s**upervised problem** because our test training has both **inputs AND outputs,** and a **regression** problem because the output is **continuous** (any real

15 https://en.wikipedia.org/wiki/Discriminative_model
16 https://www.youtube.com/watch?v=bxe2T-V8XRs
17 https://github.com/stephencwelch/Neural-Networks-Demystified (supporting code to 16)

number between 0 and 100). If we were relating letter grades it would be a **classification** since the inputs map to certain classifiers.

We need to solve **type difference** between input and output. Input is (time, time) and output is (int). Therefore we scale (**normalize**) our data to be integers between 0 and 1 (i.e. X=X/max(X) and Y=Y/max(Y)).

Now we build our NN. We know it has two inputs, and one output. We call our output layer y^1 (y estimate). Any layer between the input layer x and output layer y is called a **hidden layer**. A **DBN** is a NN with many hidden layers. In this example we use one hidden layer with three hidden units. A DNN would stack many layers together (0<l<=L layers).

In the analogy, a node is a neuron, and the connection between nodes is the synapses. The job of a **synapse** is to take the value of a neuron, weight it, and output it to the next neuron. The job of a **neuron**, is to take input from all incoming synapses, and apply an activation function. Ex $z = \sum x_i$ and $a = 1 / (1 + e^{-z})$. This example models a **non-linear** problem that simple models may miss.

Hyper parameters: are constants that establish the structure and behaviour of the network. We need to model the problem first before we start training it.
Neural Network: The NN's job instead is to learn parameters (weights for each synapse). We use **matrices** to move multiple inputs through the network. **Ex** our input is a 3 x 2 matrix [**X**] and our output is a 3 x 1 matrix [**Y**]. Each element in X needs to be multiplied by a **corresponding weight w**. We can use NMF to multiply each input in X by a weight in W through matrix factorization [$X W^1 = Z^2$ (Where Z^2 is a (3 x 3) second layer)]. Now we need to apply the **activation function** to each element in Z^2 [$f(Z^2)$]. Now we can say that $a^2 = f(z^2)$. This is called **forward propogation**. Finally we propagate a^2 our output y by applying W^2 then applying the activation function $y = f(Z^3)$.

Now the **learning** stage. To improve, we need to quantify how wrong this stage is using a **cost function**. Ex. $J = \sum 0.5 * (y – yHat)^2$ (brute force). **Training** is analogous to **Minimizing a Cost Function**. In our example, the cost is a relation between the inputs and the synapse weights. We minimize the cost by changing the weights in W^i.

Gradient Descent: Suppose we look at our **cost** by combining all of our equations. $J = \sum 0.5(y – f(f(XW_1)W_2))^2$. If we want to look at whether to raise or lower the value J, we should look at the partial derivative wrt W. J is going uphill/downhill depending on positive/negative PD. If we continue taking iterative steps until dJ/dW approaches 0, then we have **gradient descent**. What if the cost function isn't convex? We want the global min, but may get a local min instead. **Stochastic GD** is when we apply our examples one at a time (vs all at once) and can help to find the best solution among possible solutions.

Backpropogation: using the derivative of the cost function, we backpropagate the error to each synapse in our NN. See video **https://www.youtube.com/watch?v=GlcnxUlrtek** for an example.

Numerical Gradient Checking: How do we check if there are errors? How do we check our math? Rather than just accept the derivative rules we learned, we apply the rise over run definition to test the limits of our function. We compare our NG to our G to quantify just how different they are.

Training: Yann LeCun *Efficient BackProp* explores optimization techniques as applied to NN. Finding the right i/o to train our NN is also a difficult problem since if we give it a poor dataset, it will compute a very large output space and thus be impractical.

Overfitting[18]**:** In the real world, predictions can't always be perfect. Our observations on data are not the data itself, therefore it's dangerous to assume that the example inputs effectively model the real world. In real life, the input is comprised of signal (input) and noise (random variables). In our example, the noise would be test difficulty, guessing during the test, and other outside factors.

This is addressed by splitting our learning stage into training and testing. Where training is as usual, but testing is meant to simulate the real world. This helps the NN separate outlier data and noise.

How do RNN's differ?

When there is no target sequence, we can get a teaching sequence by trying to predict the next term in the input sequence. In essence, an RNN is suited towards modelling temporal sequences.

Autoregressive Models: predicts the next term from a fixed number of previous terms "delay taps." Essentially a system with memory which accounts for the previous value in the current decision.

What about ConvNets[19]:

Usually used for image processing, but suited for signal processing-classification in general.

Suppose our signal is some function of time (ex. f[n] is a 1000 sample fn). Suppose the initial layer is also very large (500 neurons). For a DNN, we would need 500,000 synapses to manage this. Let the first neuron be only connected to a limited number of inputs (first 1-5 synapses) plus bias. The second neuron will be connected to a shift of 5 synapse (2-6 synapses) and bias. The output of a layer is called a **feature map**.

ConvNets are a multi-layer NN. Each layer is able to recognize some important feature in the signal (ex. In facial recognition, one layer may look for eyes, another face shape, and another noses). The ConvNet passes the signal through many layers to extract features from the signal.

Suppose we have a ConvNet which takes a 2D array of pixels. The ConvNet decides whether the picture denotes an X or an O. Shapes are particularly difficult for this problem. A DNN might match pixel by pixel. This would fail if the shape was altered. ConvNets on the other hand match shapes within the signal. When you learn to recognize a shape by its parts (called **features**) we can match pieces of the signal through filtering. By applying a filter throughout a signal, we place a **feature map** over the signal which enumerates just to what degree that particular feature relates to this section on the signal (7:20 in the video).The stack of outputs after applying these filters is called a **convolution layer**. This one signal becomes a stack of signals.

The next technique is called **pooling**. Pick a window size and a stride (hop?), then walk the window across the filtered images. From each window, take the maximum value. This will help to shrink the image. Pooling doesn't care where that max value occurs, so the function will always pick up the max value. Thus the **conv layer** becomes a smallers stack of signals.

Normalization. In this case, 0 out any negative values. Called a **Rectified Linear Unit layer.**

---

18 https://www.youtube.com/watch?v=mYIgSq-ZWE0 (Nate Silver: Signal and Noise)

19 https://www.youtube.com/watch?v=FmpDIaiMIeA

Now we take the convolution layer, ReLU layer, and Pooling layer, place them in parallel. We can also stack these many times for greater resolution.
Now we have a **fully connected layer**. Every value has an effect on the output, arrange into a single list. Each value is fed into each possible classification. Some values will resonate with each classifications. Based on the weights, we get an average of the votes for each classification (ex. X = .92 and O = 0.52).

Learning: Where do we derive the features? How about the voting weights? A. By backpropogation. In short, Error = right answer – actual answer. Then, for each feature pixel and voting weight, adjust the weight to find the lowest error.

Hyperparameters: What are the presets in a ConvNets?

- Convolution:
  - Number of features
  - Size of features
- Pooling:
  - Window Size
  - Window Stride (Hop size?)
- Fully Connected
  - Number of Neurons.

Architectural Decisions: How many of each type of layer? In what order?
(22:40 in video gives example of applying ConvNets to sound).
Rule of Thumb: If your data is still useful after flipping your 'image,' then ConvNets won't be useful.

Further Reading: Notes from Stanford CS 231 course. Justin Johnson.
Toolkits: Caffe, CNTK, TensorFlow.

How to Construct Deep Recurrent Neural Networks ***https://arxiv.org/pdf/1312.6026.pdf***

Tutorial for modeling and generating sequences of polyphonic music with the RNN-RBM
http://deeplearning.net/tutorial/rnnrbm.html

Code for a NADE ***http://www-etud.iro.umontreal.ca/%7Eboulanni/nade.py***

Audio Chord Recognition with a Hybrid Recurrent Neural Network
http://www.eecs.qmul.ac.uk/~sss31/Pubs/ISMIR2015.pdf

Yann LeCun *Efficient BackProp*

The Signal and Noise http://www.stavochka.com/files/Nate_Silver_The_Signal_and_the_Noise.pdf

Topics for next week:

Backprogagation through time (BPTT)

Stochastic Gradient Descent

NADE

Various conditional probability distributions used in the paper.

HMM's

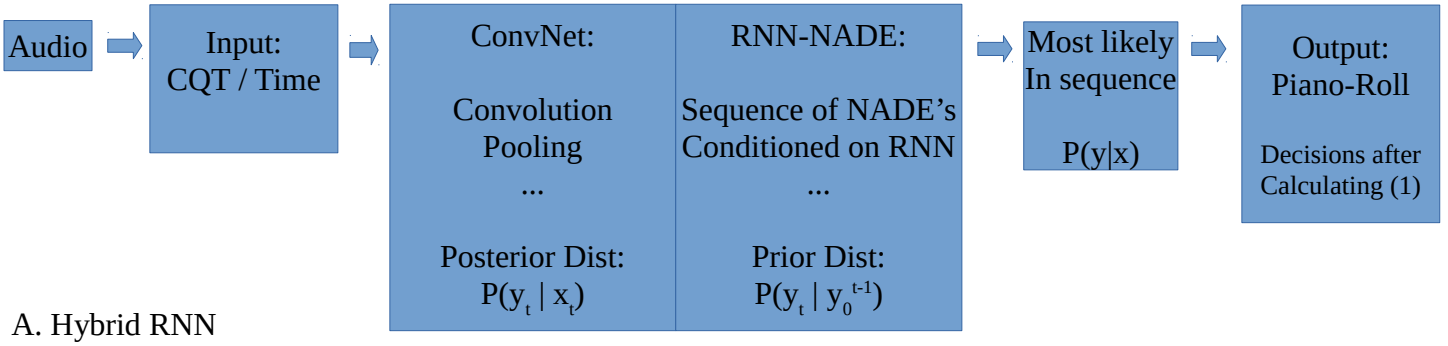
Beam Search (where does it fit in the model)

Week of Oct 16th

Summary of “An End-to-End Neural Network...” (cont)

3. Proposed Model:

(Audio) -> CQT x time -> Convolution Network -> RNN-NADE -> piano-scroll sequence



A. Hybrid RNN

Conditional probability needs to be studied more. “A hybrid RNN is similar to an HMM where the state transition properties for the HMM $P(y_t|y_{t-1})$ have been generalized to include connections from all previous outputs” (page 931).

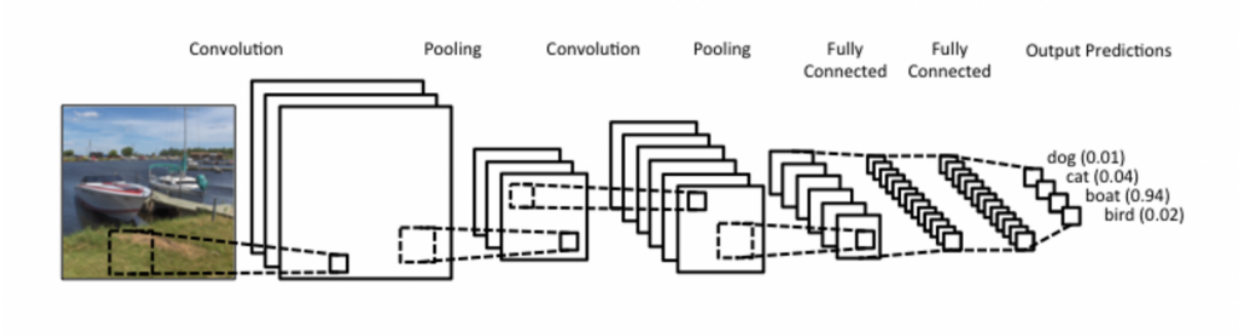
The probability for the **overall system** is determined by

$$P(y|x) \propto P(y_0|x_0)\ \Pi^{T}_{t=1}\ P(y_t|y^{t-1}{}_0)\ P(y_t|x_t) \qquad (1)$$

Where the probabilities correspond as in the diagram above.

Conjecture Training is done as per the NN example, only the output is classifier probabilities between 0 and 1 rather than output values themselves.

20



The model training is done by a cost function like the previous example $J() = \sum 0.5 * (y - yHat)^2$ where *yHat* is the NN’s probability guess and *y* is the model objectives (expected result). The best parameters are found using gradient descent.

B. Inference

20 http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/

The general structure of this model leads to a problem in decoding at test time, since the best choices for earlier choices (from 0 to t-1) hasn't been determined at time t. The proposed solution is to pick choices from the acoustic model's distribution in order of log-likelihood

How does beam search work[21]? (Optimization of best-first search)

We want to find a local maximum in a set of values (function). If the distribution is large, our search for a maximum will possibly bring us a local maximum, but not the global max. Stochastic (somewhat random) beam search considers finite number of locations at once. If one search brings better results than others, some searchers will move to that location. We want every searcher to converge, but not necessarily at every step, since they may converge too quickly. Stochastic beam search will eventually converge on the global maximum for the function.

The Hash-beam search proposed in the paper uses this model, however since the objects we are comparing are so large, we may spend too much time on similar inputs (higher density in one location, but not much change in the y axis). Therefore we hash the inputs according to a function so when we're faced with many similar choices, only look at the most probable one.

21 https://www.youtube.com/watch?v=jh7sEpy-014

Week of Oct 30th

Converting a CQT into an 88 Dimensional Representation

This time I tried to do a direct conversion of my input into output. Namely, convert a 252 dimensional CQT vector into an 88 dimensional vector that can be used as a piano-scroll representation.

In retrospect, this should be a fairly trivial process, however it took some research into CQT's before I could do the conversion with confidence. My confusion was probably in thinking of the CQT as a STFT. How do you map frequency bins into pitches? I did quite a bit of research into this. It wasn't until I read some articles that implemented CQT's that I understood how CQT bins map to frequencies.

Constant Q-Transforms frequency representation of an audio input. Whereas a Short Time Fourier Transform (STFT) is a frequency representation where each frequency bin is linearly spaced (each bin represents the same band length), a CQT has log-spaced bins. This means that CQT bins increase in bandwidth logarithmically as we go up in frequency making them well suited for music. Furthermore, the settings for the frequency bins of a CQT are specially implemented so that they correspond to musical octaves. In Librosa's CQT settings, we specify bins_per_octave, and nbins. Nbins must be an integer multiple of bins_per_octave. In reference to *An End-to-End Neural Network for Polyphonic Piano Music Transcription*, 36 bins/octave across 7 octaves will give us 252 bins.

My main confusion at this point was how 252 bins relate to 88 pitches. The answer is that it doesn't... Librosa's CQT explicitly begins the CQT process at pitch C1 and ignores the bottom three. Meaning that we have exactly 7 octaves... in other words we have 84 semitones to work with. This would imply that we have 252/3 = 84 semitones, and 3 bins per semitone. Now that I know this, the conversion process is fairly trivial.

The python files for audio conversion are: **audio_midi.py** and **audio_midi1.py** in the folder *Previous Versions*

Audio Chord Recognition With a Hybrid Recurrent Neural Network
http://www.eecs.qmul.ac.uk/~sss31/Pubs/ISMIR2015.pdf

From Sounds to Music and Emotions: 9th International Symposium CMMR 2012

https://books.google.ca/books?id=zWG5BQAAQBAJ&pg=PA309&lpg=PA309&dq=mapping+cqt%27s+to+pitches&source=bl&ots=MmTh8TulYc&sig=KgQRwHhk0V3tPD5LP5Ap0-MRKbc&hl=en&sa=X&ved=0ahUKEwiN0qilk6_QAhVSVWMKHazgD2IQ6AEIHjAA#v=onepage&q=mapping%20cqt%27s%20to%20pitches&f=false

CQT-Based Convolutional Neural Networks for Audio Scene Classification
https://www.cs.tut.fi/sgn/arg/dcase2016/documents/workshop/Lidy-DCASE2016workshop.pdf

Pitch Shifting of Audio Signals Using the Constant-Q Transform
https://www.dafx12.york.ac.uk/papers/dafx12_submission_81.pdf

Week of Nov 6th & 13th

Converting Midi into a Piano Roll

For these two weeks I attempted to convert a midi file into a piano roll representation. This proved to be more difficult than I first thought.

Earlier I had found the python library music21 which converts Midi files into ‘streams’ to operate on. The first week was spent on music21. The documentation for the library is very large so it took a while to get a grasp of the music21 objects involved, as well as the methods to access the information. Unfortunately, after a week of working with the library, I wasn’t fully able to use it for this purpose.

Music21 operates by wrapping midi information into a hierarchy of music21 objects. A midi file is read into a stream object. If applicable, it’s converted into the subtype ‘score.’ A score is comprised of ‘parts’ objects which in turn encompass ‘note’ objects. If multiple note objects occur simultaneously, then they are further wrapped in a ‘chord’ object. While this hierarchy is useful for working withing the music21 environment, it makes it difficult to access the raw data for processing outside the environment. For example, reading a midi file into a stream object may instead convert it into a score object. Unfortunately, stream and score objects differ in the way they encapsulate objects below it in the hierarchy... So if you’re using stream commands to iterate over the notes it may crash because it converted the midi file into a score without announcing it. An additional issue is in the accessing of information of notes vs chords. Sometimes simultaneously occuring notes will be wrapped in a chord object. However, the means to access note names differs between the two objects, which means iterating over the score requires multiple handling situations to access the same data. It’s possible, but not suited for my purposes. The attempts to iterate over the notes in a midi file are documented in **midi_binary.py** and **midi_binary1.py**.

The second week was spent looking for libraries that might do the conversion for me. The first promising result was the LeNet tutorial for generating polyphonic sequences (http://deeplearning.net/tutorial/rnnrbm.html). While this was promising, it was written in Python 2.7 and wasn’t quite as robust as I hoped. Later I spent time looking through academic articles. Piano Roll representations are very common in transcription problems so there must be a method documented at some point. So I looked through articles that involved transcription and followed their references until I found pretty_midi ([3] in the bibliography for report #1 below). This was the most efficient library I found, however the piano roll uses midi note numbers so the dimensionality is 128... I expect we want to reduce the dimensionality of the NN, so I implemented a method to reshape the array into 88 dimensional format.

The python file is **midi_to_roll.py** in the folder *Previous Versions.*

Week of Nov 20th, 27th & Month of Dec

Working with Keras and Sci-kit Learn

The first Week was spent working with Sklearn. This is the main machine learning library in the Scipy environment. While this library is well documented, and has a lot of options. It doesn't seem to offer robust support for multi-output classification. Since I'm working with polyphonic music, I want my output to classify more than one thing at a time.

Next I found Keras. This library is quite good, as the methods to build a NN are very intuitive. Models are built layer by layer, meaning any combination of NN's is potentially possible. It also implements every activation, and optimization I've ever come across. I was able to get a promising output on almost the first try.

Unfortunately, I underestimated how difficult it is to determine hyper parameters. I spent the first couple weeks experimenting with the hyper-parameters in Keras. Dropout layers, activation layers, callbacks and so on, as I felt that the accuracy was quite low. Initially, I had planned to implement RNN and ConvNets for comparison, but I didn't have time as the DNN alone required more work than I anticipated. Instead, I decided to experiment outside the bounds of the paper. In particular I heard a lot about Adam, so I experimented with the newer optimizers. They were actually more promising than ADADELTA, but they all performed about the same. The most noticeable difference between them was that they each had significant differences in the amount of epochs needed to run before early callbacks cut them off. Adam required around half the epochs (111/200), Adamax never stopped, so it ran the full number of epochs (200/200), and Nadam consistently ran the least at (64/200). I don't know what this would translate to in terms of performance. I looked up how Nesterov Momentum works, so it makes sense that Nadam would be faster than Adam. Clearly Nadam trains the fastest, since early callback is breaking the training early, does this translate to a difference in overfitting? Either way, Nadam trains the fastest, so it was used as the benchmark.

The initial prototyping for the framework is in **transcription_machine.py** in *Previous Versions*.
The final prototyping for the framework is in **DNN_transcription_testAudio.py** in *Previous Versions.*

The final submission for the framework is **DNN_transcription.py**

There were some things I came across towards the end that I didn't get a chance to implement:

http://machinelearningmastery.com/evaluate-performance-deep-learning-models-keras/
This page described more robust methods of evaluating the NN. In particular it combines Keras with the methods in Sklearn

http://machinelearningmastery.com/grid-search-hyperparameters-deep-learning-models-python-keras/
This would have been helpful at the beginning, but unfortunately I found it towards the end. This tutorial covers how to use sklearns grid search method to tune the NN optimally. My method was simple trial and error by trying to maximize the score.

I wanted to implement the RNN and ConvNet acoustic models in [13], but I ran out of time. It seems that the Keras requires the dataset to be of the form (m, m, n_samples), which is very different to my current dataset. Furthermore, I would need to look into how to transform the datasets anyways (how do I the data in sets of squares? Zero-padding?) so it was left as future work.

Bibliography (for report 1)

[1] Abdel-Hamid, Ossama et. al. “Convolutional Neural Networks for Speech Recognition.” IEEE/ACM Trans. Speech Audio Process., vol. 22, pp. 1533-1545, October 2014. *https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/CNN_ASLPTrans2-14.pdf* (accessed October 26th, 2016).

[2] Britz, Denny. Understanding Convolutional Neural Networks For NLP. November 7, 2015. http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/ (accessed October 18th, 2016).

[3] Colin Raffel and Daniel P. W. Ellis. Intuitive Analysis, Creation and Manipulation of MIDI Data with pretty_midi. In 15th International Conference on Music Information Retrieval Late Breaking and Demo Papers, 2014.

[4] Convolutional Neural Networks (LeNet). *http://deeplearning.net/tutorial/lenet.html* (accessed October 5th, 2016).

[5] Deep Belief Networks. *http://deeplearning.net/tutorial/DBN.html* (accessed October 3rd, 2016).

[6] G. E. Poliner and D. P. Ellis, “A discriminative model for polyphonic piano transcription,” *EURASHIP J. Appl. Signal Process.*, vol. 207 no. 1, pp. 154-154, 2007. *https://www.ee.columbia.edu/~dpwe/pubs/PoliE06-piano.pdf* (Accessed October 28th, 2016).

[7] Ghahramani, Z. (2001) An Introduction to Hidden Markov Models and Bayesian Networks. International Journal of Pattern Recognition and Artificial Intelligence. 15(1):9-42.

[8] H. Larochelle and I. Murray, “The neural autoregressive distribution estimator,” in Proc. Int. Conf. Artif. Intell. Statist., 2011, pp. 29-37.

[9] Larochelle, Hugo. *Neural networks [9.3]: Computer vision – parameter sharing*. (Video) November 15, 2013. *https://www.youtube.com/watch?v=aAT1t9p7ShM* (accessed October 4th, 2016).

[10] P. Smaragdis and J. C. Brown, “Non-negative matrix factorization for polyphonic music transcription,” in *Proc. IEEE Workshop APPll. Signal Process. Audio Acoust.,* 2003, pp. 177-180. http://www.ee.columbia.edu/~dpwe/e6820/papers/SmarB03-nmf.pdf (Accessed October 28th, 2016).

[11] Rohrer, Brandon. *How Convolutional Neural Networks work.* August 18th, 2016. *https://www.youtube.com/watch?v=FmpDIaiMIeA* (accessed October 14th, 2016).

[12] Rojas Raul. *Neural Networks – A Systematic Introduction*. Springer-Verlag, Berline, New-York, 1996. *https://page.mi.fu-berlin.de/rojas/neural/* (accessed October 6th, 2016).

[13] Siddharth Sigtia, Emmanuiol Benetos, and Simon Dixon An End-to-End Neural Network for Polyphonic Piano Music Transcription,” in *Proc. IEEE Workshop APPll. Signal Process. Audio Acoust.*, vol. 24, no. 5, pp. 927-940, May 2016.

[14] Silver, Nate. “The Signal and the Noise.” | Talks at Google. November 28, 2012. *https://www.youtube.com/watch?v=mYIgSq-ZWE0* (accessed October 11th, 2016).

[15] Simpson, Andrew J.R. “Deep Karaoke: Extracting Vocals from Musical Mixtures Using a Convolutional Deep Neural Network.” April 17, 2015. https://arxiv.org/ftp/arxiv/papers/1504/1504.04658.pdf (accessed September 28th, 2016).

[16] Srivasta Nitish, Geoffrey Hinton et al. *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*. Journal of Machine Learning Research (2014). Published April 2014. *https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf* (accessed October 4th, 2016).

[17] Welch, Steven. *Neural Networks Demystified.* (Video) November 4th, 2014. *https://www.youtube.com/watch?v=bxe2T-V8XRs* (accessed October 2nd, 2016).

[18] Welch, Steven. *Neural Networks Demystified.* (Code) Latest Commit August 23, 2016. https://github.com/stephencwelch/Neural-Networks-Demystified (accessed October 2nd, 2016).

[19] Weston, Jason. *Support Vector Machine (and Statistical Learning Theory) Tutorial.* Last Modified 2004. *http://www.cs.columbia.edu/~kathy/cs4701/documents/jason_svm_tutorial.pdf* (accessed October 6th, 2016).

[20] Meinard Müller, Verena Konz, Wolfgang Bogler, and Vlora Arifi-Müller **Saarland Music Data (SMD)**
In Late-Breaking and Demo Session of the International Conference on Music Information Retrieval (ISMIR), 2011.